Practical Applications of Artificial Intelligence in Software Testing

Abhishek T, Akshayavarshini S, Narendra Babu M, Prarthana K, Vaishnavi S

Submitted to School of Computer Science and Engineering,
Vellore Institute of Technology, Chennai 600127 India

Abstract: In this paper, we analyze the practical applications of artificial intelligence in each phase of software testing. The advantages and disadvantages of automated testing using AI and Machine Learning (ML) are analyzed. Then, practical applications and the current trends of automated testing is discussed. The main goal is to make insights about what can be done in different stages of software testing utilizing AI and to explore the areas of expansion for AI-based software testing.

Index Terms: Software Testing, Automated Testing, Machine Learning, Artificial Intelligence

1. Introduction

Software applications today are rapidly increasing. There is a great deal of competition among stakeholders in terms of time, cost and scope. The fundamental aspect of automating the testing process is to reduce the time, in addition to scope. In case of a highly dynamic software which demands immediate response to rapid changes, it is necessary to take quick actions against gaps introduced by complexity and fast changes in testing cycles. At this point, one of the most feasible technologies, Machine-Based Intelligence is used for testing. The key pillars of AI that can be used for software testing include Machine Learning(ML), deep learning and natural language processing (NLP). AI can enhance testing quality, generate smart and accurate test cases for systems and enhance the testing coverage.

2. Motivation

As the complexity of software continues to increase and the release delivery cycles become shorter, testers need to provide quality feedback to developers nearly instantaneously. Thus for continuous testing and delivery, artificial intelligence is one key factor for efficient testing. For example let us consider a website to be tested and let's say the developer changes the ID, name or some attribute of an element on the web page. Once the test is run, it immediately breaks due to this change because it is still referring to the old ID. This leads to unstable test and the tester has to modify and spend a lot of time in maintaining that test for all the changes to be done.

This led to the introduction of the concept of dynamic locators in software testing. The artificial intelligence underneath the platform in real-time analyzes all the DOM objects of the page or software and extracts the object along with its properties. Based on this analysis, the AI uses the best strategy to locate a particular element. When using artificial intelligence, even if a developer changes the attribute of an element, the test continues to run and this leads to a more stabilized testing. As a result of this, authoring, execution and maintenance of tests are much faster and more stable.

Thus the motivation for this work is to reduce manual effort in software testing by introducing several automation processes into projects and to increase the efficiency of the testing process

using Artificial Intelligence (AI) and Machine Learning (ML).

3. Related Work

Test Automation uses various ML algorithms and techniques like Training Dataset to learn and predict results. The differences between algorithms used in testing are mostly the mathematical and statistical models, workflows, accuracy and ability to classify or rectify bugs. While reading about previous works in this field we identified various techniques that were used for the integration and automation for the testing process, of which a few are discussed here.

One of the proposed solutions highlighted in Briand, Labiche and Bawar's Research article is called machine Learning-based refinement of BlackBox test specification (MELBA). MELBA is a semiautomated iterative methodology based on the C4.5 algorithm. The C4.5 algorithm is used to generate decision trees in ML. The results showed that it was quite successful in fault detection although an increase in test size was needed.

Self-healing and Self-testing are the most important features of automated testing for industries. Tariq, Jason, Wendy(2019) and other researchers conducted a study on Artificial Intelligence for Software Testing (AIST) from an industry perspective. They concluded that AIST is an emerging field aimed at developing AI tools to test software and has more scope for work and research.

4. AI in Testing

Compared to humans, machines decide much faster. At least for the rough estimations, AI results can provide quick feedback. AI can be utilized in each stage of testing. [1] In the Test Definition phase, AI ensures the product quality and maximum coverage of Use cases. AI analyses and builds a model on the system by performing predictions using cause and effect. Thus, expected inputs parameters are learnt. In the implementation phase, AI with the given Inputs and the Outputs to be generated it can predict the intermediate operation that needs to be performed on the Input to get the desired output, thus it could generate various patterns with which auto generating of code can be made possible.

4.1. Why do we need AI in testing?

AI testing is so relevant today, as the enterprises around us are accelerating their digital efforts and embrace the DevOps and Agile [2], [3], [4]. But, testing and QA organisations are not equipped yet to adapt to the industry 4.0. The loopholes and bugs in the software are to be found as soon as possible to avoid cyber crimes and data thefts by hackers. AI software should be tested at system and functional levels. The Quality of service (QoS) parameters are validated as it is validated for conventional system testing. AI testing necessarily focuses on their unique features like learning capability, timeliness, etc., when compared to conventional software testing. AI makes testing more efficient and smart. As AI testing saves time, the time spent on manual testing can be used to focus on other innovative and complex tasks. AI based testing increases readiness for automation, removes duplicates and identifies reusability in codes. It also maintains optimal test coverage. [1]

4.2. ML and AI Techniques

The main goal of our ongoing research is to provide a specific set of guidelines for automating software testing processes with the aid of AI and ML techniques. With machine learning we can define the goals for testing, acquire the necessary knowledge about the system that is needed to test, design and specify test scenarios in a more efficient manner, write test automation scripts and classify the test to be done manually , analyse the result generated.

4.3. Regression

Regression may be a linear approach for modeling the connection between two variables. The variable , "y", is that the quantity we might wish to predict. We predict the variable using the

known value of the experimental variable , “x”. The goal of straightforward rectilinear regression is to make a function that takes the experimental variable as input and outputs a prediction for the worth of the variable . A linear relationship is used frequently by statistical researchers to predict the (average) numerical value of Y for a given value of X employing a regression line. If you recognize the slope and therefore the y-intercept of that regression curve , then you’ll connect a worth for X and predict the typical value for Y. In other words, you are expecting (the average) Y from X. We can use the Test Cases to coach the system with Machine learning algorithms and generate a rectilinear regression . Using this regression method, we will predict the output of the given testcase with the correlation between input parameters.

5. Experimental Details

The experimental set up is to illustrate the working model of Automated software Testing with sample testcases and their respective outputs.

5.1. Methodology

We have considered a simple office application with minimal functions to carry out an experiment with Python [5] for test case generation which takes inputs and compares the actual and expected output. The Agenda here include computation of basic pay using Automated Testing strategies. For the purpose of experimental implementation, A sample data from an excel sheet is imported using python libraries; The data also contains expected output for each test case. The piece of code for testing will be executed using the data imported as input. The output will be compared with the expected output and the test case will be determined as pass or fail; The result is saved in the excel sheet. A line graph is generated which has Test Case ID on the x-axis and the output generated on the y-axis. The generated graph will compare the expected output and actual output. The python libraries used for this experiment are described below.

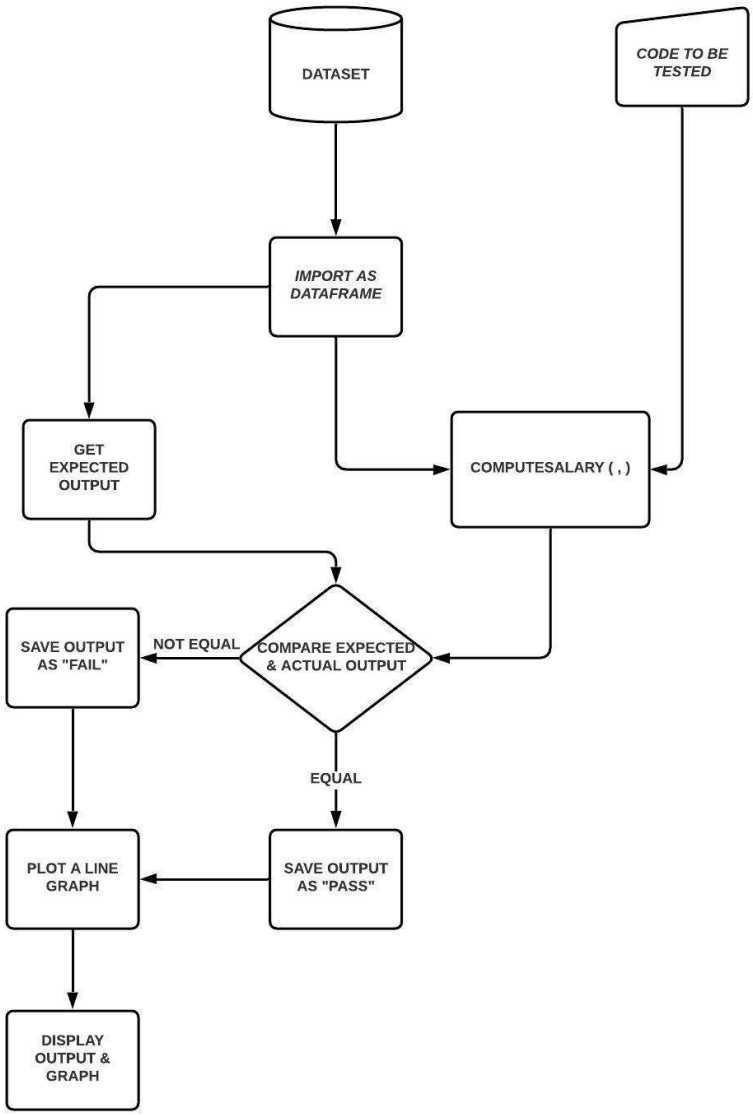
- **Pandas**
  Pandas is a library written for python, whose function is data analysis and manipulation. In other words, it manipulates numerical tables using data structures and operations.
- **NumPy**
  NumPy is a python library which deals with multi-dimensional, large arrays and matrices. It also has a huge collection of high-order mathematical functions for operating the arrays.
- **Pyplot** This is a plotting library of python. Matplotlib.pyplot has a collection of command-line style functions which makes the matplotlib work like MatLab. This is the library used for the plotting in the experimental code.

5.1.1. Experimental Framework

The experimental framework consists of three phases namely data collection, implementation using python and analysis of results. These phases are described in the following sections.

1) **Data Collection**
   In this phase the actual data collection takes place. A list of attributes, testing time, complexity measures, functional requirements and test cases will be constructed. A dataset consisting of test ID, parameters, expected output with actual output and result is constructed. All the data collected will be displayed in an excel sheet for easy access in the implementation part.
2) **Implementation**
   Using this input set, the python program will be executed for each test case. Thus the output for each test case is collected. Based on the comparison with the expected and actual output values the Result is classified as “FAIL” or “PASS” accordingly.
3) **Analysis**
   After the execution of code, a table displaying actual and expected output is shown along with the result column. If the values in actual and expected output columns match “PASS” will be

displayed as a result, else “FAIL” will be displayed. Small point difference is considered here. A line graph containing test case ID on x-axis and output range on the y-axis is generated. If the data points on the graph coincide with the lines at every instance of the test cases, then the code has passed all the test cases.



Fig(1) Flowgraph of the Testing Process

6. Experimentation Result

6.1. Dataset used

	A	B	C	D	E	F
1	Test ID	Parameter1	Parameter2	Expected Output	Actual Output	Result
2	1	500	8	662.5		
3	2	400	12	795		
4	3	525	26	-1		
5	4	350	10	695		
6	5	550	6	728		
7	6	350	7	463.75		
8	7	400	0	-1		
9	8	350	-5	-1		
10	9	600	14	1192.5		
11	10	750	8	993.75		
12	11	600	6	795		
13	12	850	-2	-1		
14	13	575	9	1142.5		
15						

6.2. Implementation

The following is the implementation of Software Testing using Machine Learning with Python

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#%matplotlib inline
```

The file containg TestCases is readed

```
df1 = pd.read_csv("d:\\TestCases.csv")
```

This is the piece of code to be tested

```
def ComputeSalary(basic,hours):
    if (hours > 0 and hours <= 24):
        if(hours > 8):
            basic = basic*1.5
        da=basic*0.25
        hra=basic*0.15
        pf=(basic+da)*0.12
        ta=basic*0.075
        netpay=basic+da+hra+ta
        grosspay=netpay-pf
        return grosspay
    else:
        return -1
```

Execution

```
for i in range(0,len(df1)):
    df1["Actual Output"][i] =␣
 ↪ComputeSalary(df1["Parameter1"][i],df1["Parameter2"][i])

df1.loc[df1["Actual Output"] == df1["Expected Output"],"Result"] = "PASS"
df1.loc[df1["Actual Output"] != df1["Expected Output"],"Result"] = "FAIL"
```

Graph Plot

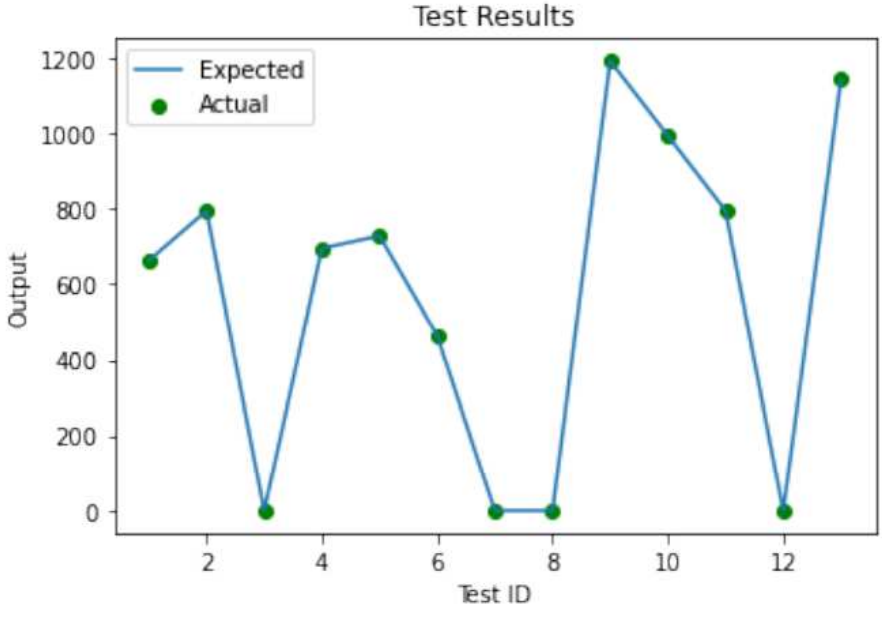
```
plt.title("Test Results")
plt.xlabel("Test ID")
plt.ylabel("Output")
plt.plot(df1["Test ID"], df1["Expected Output"])
plt.scatter(df1["Test ID"], df1["Actual Output"],color = "green")
plt.legend(['Expected','Actual'])
plt.show()
```

6.3. Output

Tabular Representation

	Test ID	Parameter1	Parameter2	Expected Output	Actual Output	Result
0	1	500	8	662.50	662.5000	PASS
1	2	400	12	795.00	795.0000	PASS
2	3	525	26	-1.00	-1.0000	PASS
3	4	350	10	695.00	695.6250	FAIL
4	5	550	6	728.00	728.7500	FAIL
5	6	350	7	463.75	463.7500	PASS
6	7	400	0	-1.00	-1.0000	PASS
7	8	350	-5	-1.00	-1.0000	PASS
8	9	600	14	1192.50	1192.5000	PASS
9	10	750	8	993.75	993.7500	PASS
10	11	600	6	795.00	795.0000	PASS
11	12	850	-2	-1.00	-1.0000	PASS
12	13	575	9	1142.50	1142.8125	FAIL

Graphical Representation



7. Discussion

Testing is a critical task in the software development process and imposes time and cost constraints on the development process. Therefore, testing performance has been increased by automating the testing process. Usage of testing activities has lots of advantages. Firstly, test coverage is improved by means of AI. Machines learn system patterns and tests are generated

automatically. Compared to humans, machines decide much faster which is proved as AI applications provide extra speed in all stages of testing and also provide quick feedback. With ML algorithms, the risk to miss bugs and cost is minimized with early fixes. Some sample works of ML in ST have been reviewed and classified. This provides us with the opportunity to extract the prominent information from existing works in ML and ST. In testing, advantages of AI applications are unneglectable. In short we can conclude that AI is a safe and beneficial tool for testing.

8. Future Work

Software will continue to be a part of human life as far as the future can possibly exist. Machine Learning has made some great hurdles in other domains of computer science and recently it has been a witness to these techniques being used in software testing also. AI comes up with more tests, much faster and more accurate. The primary goal is to achieve "Quality at speed" and faster results. Testing thousands of regression test cases that can take hours and sometimes days. So we expect more pertinent approaches. That is where AI and ML shows its power. We have included test cases for automated testing using Machine Learning with python. We have used Juypter Notebook tools to implement similar test cases [6]. As a future work, maximum usage of ML and AI algorithms will be included in the software testing process. This will help testers to choose the most efficient and appropriate learning methods for automation of a target testing stage. Data visualization is representing and visualizing the data graphically by using visual elements like charts, maps and graphs. This provides an accessible way to understand the patterns used in data. Using ML algorithms we get more refined visual representation of data.

9. Challenges and Needs

AI-based software testing is still under the process of research and many features of it are still undiscovered. Yet automated testing is deployed in several industries and is seeing phenomenal growth in the software development life cycle. Thus AI plays a crucial role in all aspects of software development and thus has a few important challenges that arise when using AI for software testing are as follows: The increasing usage of AI-based testing in software applications brings up quality assurance concerns. Most of the AI models are developed to test functionalities and behaviours of the software system but do not support much in the aspect of quality assurance. The existing quality assurance standards help in validating only 2 aspects of the entire system which are functional quality and non-functional quality parameters. The functional parameters include the program execution, behaviour, decision-making process, input-output pairing and other operations. Non-functional quality parameters are the performance scalability, security reliability, availability etc. When an AI-based program is used for testing a new quality assurance need arises which should measure the accuracy, consistency, relevancy and correctness of the AI system involved in the testing. With the growth of technology in the past few decades, several automation tools have been developed to assist engineers and software testers in the process of software development. These tools again focus only on the functional and non-functional parameters as mentioned above. Most of these tools do not have automatic validation methods and solutions for AI software systems.Thus the challenges mentioned above the pave way to the future works and requirements for AI-based software testing which include

- Development of a well-defined validation model for AI functions and features that include object detection and classification, prediction features and so on.
- Developing a well defined Quality Assurance program and establishing standards to measure the accuracy, consistency and correctness of the AI system involved in testing
- Developing solutions and tools to address the special needs and features of AI software applications.

10. Conclusion

Using ML and AI reduces the cost and accelerates the development and delivery speed, and that's what companies want to stay ahead in their competition. ML makes testing smarter and of course, It's the age of machine learning. Companies are using this technology to combat risk, enhance sales, block fraud, and to streamline manufacturing. In today's state of affairs where companies are stepping into Rapid Development, there arises a time constraint, Quality is critical, so is Speed. By leveraging the power of ML in software development, testers get additional time to test different use cases and get instantaneous feedback. Having analyzed the blessings of using AI and ML in software testing it's just a matter of developing efficient testing methodologies by allocating the right resources, the usage of reliable algorithms, and lowering manual labour only to commercial enterprise tactics that AI can't capture.

References

[1] H. Hourani, A. Hammad, and M. Lafi, "The impact of artificial intelligence on software testing," in *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*. IEEE, 2019, pp. 565–570.

[2] J. Gao, C. Tao, D. Jie, and S. Lu, "What is ai software testing? and why," in *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE, 2019, pp. 27–2709.

[3] A. M. Nascimento, L. F. Vismari, P. S. Cugnasca, J. B. C. Júnior, and J. R. de Almeira Júnior, "A cost-sensitive approach to enhance the use of ml classifiers in software testing efforts," in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. IEEE, 2019, pp. 1806–1813.

[4] T. M. King, J. Arbon, D. Santiago, D. Adamo, W. Chin, and R. Shanmugam, "Ai for testing today and tomorrow: industry perspectives," in *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE, 2019, pp. 81–88.

[5] D. Kuhlman, *A python book: Beginning python, advanced python, and python exercises*. Dave Kuhlman Lutz, 2009.

[6] C. Rossant, *IPython Interactive Computing and Visualization Cookbook: Over 100 hands-on recipes to sharpen your skills in high-performance numerical computing and data science in the Jupyter Notebook*. Packt Publishing Ltd, 2018.